

\*) Use the language to calculate the size of an object

Eg 1: #define NO\_ELEM (array) (size of array / size of array[0]);

Eg 2: node = (NODE \*) malloc (size of (struct node));

\*-> Comments

\*) Don't belabour the obvious

Eg 1: i++; // Increment the value of i

Eg 2: const float PI = 3.1415 // Initialise PI as a constant

\*) Comment functions and global data

Eg 1:   
:   
int rear = -1; // Rear end of the queue   
:

Eg 2: void push (int ele) // Pushes an element to the   
{   
top of stack.   
}

\*) Don't comment bad code, rewrite it

Eg 1: /\* if result is zero means match found so return   
(true). Otherwise no match found means

return false (zero) \*/   
result = strcmp (str1, str2)   
if (result == 0)   
return ! (result);

else   
return ! (result);

Eg 2) /\* Ends on points to a pointer to character \*/   
char \*\*p; \*\*p = 'A';

\*) Be Accurate

Eg 1: `for (i=0; i <= n; i++)` // i < n as array elements are indexed from 0 to n-1  
`cin >> a[i];`

Eg 2: #define isnum(n) (num >= '1' && num <= '9') // if num >= '0' is accurate)

## EXPRESSIONS AND STATEMENTS

\*) Indent to show structure

Eg 1: `for (i=0; i < n; i++)`  
`for (j=0; j < n; j++)`  
`cin >> a[i][j];`

; // Clean indentation shows which statement is a part of which loop \* 1

Eg 2: `switch (ch)`  
`{`  
`case 1: A = p1 * n1 * n1;`  
`break;`  
`case 2: A = l * b;`  
`break;`  
`}`

\*) Use the natural form of expressions

Eg 1: `if (x != 10 && y != 20)` // (De Morgan's Law) for  
// `!(x == 10 || y == 20)`

Eg 2: `if (!true);` // Use `if (false)`

Eg 2: // Computes the product of two integers and returns their product + 1

```
float product (float p, float q);
```

\*) Use Meaningful Names

Eg 1: #define PI 3.1415

Eg 2: #define Frame Size 5

\*) Use descriptive names for globals and short names for locals

Eg 1: int Queue[50], rear = -1, front = 0;

```
void Insert (int ele) // Here ele is the new element
{
```

```
    rear ++;
```

```
    Queue[rear] = ele;
```

```
}
```

Eg 2: #define PI 3.1415

```
float area () (float r)
{
```

```
    CArea = PI * r * r;
```

```
}
```

## STYLES

### I) NAMES

#### \* Choose good names

Eg 1:  $Area = PI * radius * radius;$

/\* Here the variable names are self explanatory.

Eg 2:  $q[rear] = newElement;$

// Inserting element to the rear end of the Queue

#### \* Keep comments in synch

Eg 1: /\* Computes area of circle and triangle \*  
switch (ch)

{

case 1:  $area = PI * r * r;$   
break;

case 2:  $area = 0.5 * b * h;$   
break;

case 3:  $area = l * b;$   
break;

// Area of rectangle - not updated in comment.

}



COMMA SEPARATED VALUES

Eg 1: "Software Testing", "6", "CS6", "NMAHIT"  
Eg 2: "New York", "16/01/2018", "28", "27.444", "27.32"

### \* ) Parenthesize to resolve Ambiguity

Eg 1: if ( (x=100) && (y=20) ) // helps to understand  
{ // the test condition correctly  
:  
}

Eg 2: p++ = (p-1) \* (q-1); // if ( ) is not used then  
// \* takes higher precedence  
// resulting in inaccurate result

### \* ) Break up complex expression

Eg 1: temp = (b\*b - 4\*a\*c);  
temp = -b + sqrt(temp);  
temp = temp / (2\*a);  
x = temp //  $x = \frac{-b + \sqrt{b^2 - 4ac}}{2a}$

Eg 2: p += ( q = (a < 10) ? z : w );

can be broken down as

if (a < 10)

q = z;

else

q = w;

p = q;

### \* ) Be Clear

Eg 1: max = (a > b) ? a : b

can be stated more clearly as

if (a > b)

max = a;

else

max = b;

\* Don't contradict the code

Eg 1: Find out value of number +1

```
if (num >= 0 && num <= 10)
```

```
{
```

```
:
```

```
}
```

Eg 2: Return true if equal else false

```
if (strcmp (str1, str2))
```

```
return 0;
```

```
return 1;
```

\* Clarify, don't confuse

Eg 1: strcmp returns -1 if s1 is above s2 in ascending order else 0 if equal else +1

```
int strcmp (char *s1, char *s2)
```

```
{
```

```
:
```

```
}
```

// This is confusing, we can just say it returns

```
0 if s1 == s2
```

```
-1 if s1 < s2
```

```
1 if s1 > s2
```

Eg 2: max is assigned with the value<sup>0</sup> if the expression returns false else it assigns 1

```
max = (a > b) ? 1 : 0
```

// In simple words if (a > b) max = 1 else max = 0

\*)

### Conventions

Eg 1: 

```
class OddThread
{
    ...
}
```

 // In JAVA, the convention for class names is using camel case.

Eg 2: 

```
const int PI = 3.1415;
```

 // Capital letter for constants

### \*) Use Namespaces

Eg 1: 

```
struct Queue
{
```

```
    int n; // n instead of number of elements
    int front; // front instead of Front of Queue
    int rear
    int q[50];
};
```

Eg 2: 

```
class AreaOfCircle
{
```

```
    int rad; // r instead of radius of circle
    int area; // and area of circle +/
    :
};
```

### \*) Use Active name for Functions

Eg 1: 

```
if (isack (var)) ...
```

Eg 2: 

```
if (isprime (num)) ...
```



Eg 2: `flag = (cost < 50) ? 0 : 1`  
 can be written as  
`if (cost < 50)`  
`flag = 0;`  
`else`  
`flag = 1;`

\*) Be careful with side effects

Eg 1: `scanf ("%d %d", &y1, &loss[y1])`  
 may produce errors. Instead we may write as

`scanf ("%d", &y1);`  
`scanf ("%d", &loss[y1]);`

Eg 2: `arr[i++] = i;`  
 can be written as  
`arr[i] = i;`  
`i++;`

\*) CONSISTENCY AND IDIOMS

\*) Using consistent indentation & brace style

Eg 1: `if (country == FBA)`  
`{`  
`:`  
`}`  
`else`  
`{`  
`:`  
`}`

Eg 2: `for (int i = 0; i < n; i++)`  
`{`  
`cout << endl;`  
`for (j = 0; j < n; j++)`  
`{`  
`cout << arr[j];`  
`}`  
`}`

## → \*) FUNCTION MACROS

### \*) Avoid function macros

Eg 1: `#define SQUARE(x) (x) * (x)` // These must  
Eg 2: `#define AREA(x) (3.14 * (x) * (x))` // be avoided;  
// use inline functions instead

### \*) Parenthesize the macro body and function

Eg 1: `#define cube(x) ((x) * (x) * (x))`  
Eg 2: `#define RADIAN(l, b) ((l) * (b))`

## → \*) MAGIC NUMBERS

### \*) Give Names to Magic Numbers

Eg 1: `#define MINROW 1`  
Eg 2: `#define MAX_LIMIT 999`

### \*) Define numbers as constants, not Macros

Eg 1: `const int PI = 3.1415;`  
Eg 2: `const int MINROW = 1;`

### \*) Use character constants not integers

Eg 1: `if (ans == 'Y' || ans == 'y') ....`  
Eg 2: `if (c >= 'A' && c <= 'Z') ..`

a) Use idiom for consistency

Eg 1. 

```
for (i=0; i<n; i++)
```

 // Conventional form of for loop  
{  
:  
}

Eg 2. 

```
while (1)
```

 // Conventional form of an infinite loop  
{  
:  
}

a) Use else-if for multiway decision.

Eg 1. 

```
if (ch == '1')
```

 // Here if-else ladder  
{ is used for  
A = P \* 2 \* 3; multiway decision  
}  
else if (ch == '2')  
{  
A = 6 \* b;  
}  
else  
{  
A = 13.5 \* b \* h;  
}

Eg 2. 

```
if (month == "FEB")
```

  
days = 28;  
else if (month == "MAR")  
days = 31;  
else  
days = 30;

## STYLES

### I) NAMES

#### \* Choose good names

Eg 1:  $Area = PI * radius * radius;$

/\* Here the variable names are self explanatory.

Eg 2:  $q[rear] = newElement;$

// Inserting element to the rear end of the Queue

#### \* Keep comments in synch

Eg 1: /\* Computes area of circle and triangle \*  
switch (ch)  
{

case 1:  $area = PI * r * r;$   
break;

case 2:  $area = 0.5 * b * h;$   
break;

case 3:  $area = l * b;$   
break;

// Area of rectangle - not updated in comment.  
}



Eg 2: // Computes the product of two integers and returns their product +1

```
float product (float p, float q).
```

\*) Use Meaningful Names

Eg 1: #define PI 3.1415

Eg 2: #define Frame Size 5

\*) Use descriptive names for globals and short names for locals

Eg 1: int Queue[50], rear = -1, front = 0;

```
void Insert (int ele) // Here ele is the new element
{
```

```
    rear ++;
```

```
    Queue[rear] = ele;
```

```
}
```

Eg 2: #define PI 3.1415

```
float area () (float r)
{
```

```
    CArea = PI * r * r;
```

```
}
```

\*)

### Conventions

Eg 1: 

```
class OddThread
{
    ...
}
```

 // In JAVA, the convention for class names is using camel case.

Eg 2: 

```
const int PI = 3.1415;
```

 // Capital letter for constants

### \*) Use Namespaces

Eg 1: 

```
struct Queue
{
```

```
    int n; // n instead of number of elements
    int front; // front instead of Front of Queue
    int rear
    int q[50];
};
```

Eg 2: 

```
class AreaOfCircle
{
```

```
    int rad; // r instead of radius of circle
    int area; // and area of circle +/
    ...
};
```

### \*) Use Active name for Functions

Eg 1: 

```
if (isack (var)) ...
```

Eg 2: 

```
if (isprime (num)) ...
```

\*) Be Accurate

Eg 1: `for (i=0; i <= n; i++)` // i < n as array elements are indexed from 0 to n-1  
`cin >> a[i];`

Eg 2: #define isnum(n) (num >= '1' && num <= '9') // if num >= '0' is accurate)

## EXPRESSIONS AND STATEMENTS

\*) Indent to show structure

Eg 1: `for (i=0; i < n; i++)`  
`for (j=0; j < n; j++)`  
`cin >> a[i][j];`

; // Clean indentation shows which statement is a part of which loop \* 1

Eg 2: `switch (ch)`  
`{`  
`case 1: A = p1 * 2 * pi;`  
`break;`  
`case 2: A = l * b;`  
`break;`  
`}`

\*) Use the natural form of expressions

Eg 1: `if (x != 10 && y != 20)` // (De Morgan's Law) for  
// `!(x == 10 || y == 20)`

Eg 2: `if (!true);` // Use `if (false)`

### \* ) Parenthesize to resolve Ambiguity

Eg 1: if ( (x=100) && (y=20) ) // helps to understand  
{ // the test condition correctly  
:  
}

Eg 2: p++ = (p-1) \* (q-1); // if ( ) is not used then  
// \* takes higher precedence  
// resulting in inaccurate result

### \* ) Break up complex expression

Eg 1: temp = (b\*b - 4\*a\*c);  
temp = -b + sqrt(temp);  
temp = temp / (2\*a);  
x = temp //  $x = \frac{-b + \sqrt{b^2 - 4ac}}{2a}$

Eg 2: p += ( q = (a < 10) ? z : w );

can be broken down as

if (a < 10)

q = z;

else

q = w;

p = q;

### \* ) Be Clear

Eg 1: max = (a > b) ? a : b

can be stated more clearly as

if (a > b)

max = a;

else

max = b;



Eg 2: `flag = (cost < 50) ? 0 : 1`  
 can be written as  
`if (cost < 50)`  
`flag = 0;`  
`else`  
`flag = 1;`

\*) Be careful with side effects

Eg 1: `scanf ("%d %d", &y1, &loss[y1])`  
 may produce errors. Instead we may write as

`scanf ("%d", &y1);`  
`scanf ("%d", &loss[y1]);`

Eg 2: `arr[i++] = i;`  
 can be written as  
`arr[i] = i;`  
`i++;`

\*) CONSISTENCY AND IDIOMS

\*) Using consistent indentation & brace style

Eg 1: `if (country == FBA)`  
`{`  
`:`  
`}`  
`else`  
`{`  
`:`  
`}`

Eg 2: `for (int i = 0; i < n; i++)`  
`{`  
`cout << endl;`  
`for (j = 0; j < n; j++)`  
`{`  
`cout << arr[j];`  
`}`  
`}`

a) Use idiom for consistency

Eg 1. 

```
for (i=0; i<n; i++)
```

 // Conventional form of for loop

```
}
```

Eg 2. 

```
while (1)
```

 // Conventional form of an infinite loop

```
{
```

```
}
```

a) Use else-if for multiway decision.

Eg 1. 

```
if (ch == '1')
```

```
{
```

```
    A = P1 * m * n;
```

```
}
```

```
else if (ch == '2')
```

```
{
```

```
    A = lab;
```

```
}
```

```
else
```

```
{
```

```
    A = 13.5 * b * h;
```

```
}
```

// Here if-else ladder  
is used for  
multiway decision

Eg 2. 

```
if (month == "FEB")
```

```
    days = 28;
```

```
else if (month == "MAR")
```

```
    days = 31;
```

```
else
```

```
    days = 30;
```

## → \*) FUNCTION MACROS

### \* Avoid function macros

Eg 1: `#define SQUARE(x) (x) * (x)` // These must  
Eg 2: `#define AREA(x) (3.14 * (x) * (x))` // be avoided;  
// use inline functions instead

### \* Parenthesize the macro body and function

Eg 1: `#define cube(x) ((x) * (x) * (x))`  
Eg 2: `#define RADIAN(l, b) ((l) * (b))`

## → \*) MAGIC NUMBERS

### \* Give Names to Magic Number

Eg 1: `#define MINROW 1`  
Eg 2: `#define MAX_LIMIT 999`

### \* Define numbers as constants, not Macros

Eg 1: `const int PI = 3.1415;`  
Eg 2: `const int MINROW = 1;`

### \* Use character constants not integers

Eg 1: `if (ans == 'Y' || ans == 'y') ....`  
Eg 2: `if (c >= 'A' && c <= 'Z') ..`

\*) Use the language to calculate the size of an object

Eg 1: #define NO\_ELEM (array) (size of array / size of array[0]);

Eg 2: node = (NODE \*) malloc (size of (struct node));

\*-> Comments

\*) Don't belabour the obvious

Eg 1: i++; // Increment the value of i

Eg 2: const float PI = 3.1415 // Initialise PI as a constant

\*) Comment functions and global data

Eg 1:   
int rear = -1; // Rear end of the queue

Eg 2: void push (int ele) // Pushes an element to the  
top of stack.  
{

\*) Don't comment bad code, rewrite it

Eg 1) /\* if result is zero means match found so return  
(true). Otherwise no match found means

return false (zero) \*/  
result = strcmp (str1, str2)

if (result == 0)

return ! (result);

else

return ! (result);

Eg 2) /\* Ends on points to a pointer to character \*/  
char \*\*p; \*\*p = 'A';



COMMA SEPARATED VALUES

Eg 1: "Software Testing", "6", "CS6", "NMAHIT"  
Eg 2: "New York", "16/01/2018", "28", "27.444", "27.32"

\* Don't contradict the code

Eg 1: Find out value of number +1

```
if (num >= 0 && num <= 10)
```

```
{
```

```
:
```

```
}
```

Eg 2: Return true if equal else false

```
if (strcmp (str1, str2))
```

```
return 0;
```

```
return 1;
```

\* Clarify, don't confuse

Eg 1: strcmp returns -1 if s1 is above s2 in ascending order else 0 if equal else +1

```
int strcmp (char *s1, char *s2)
```

```
{
```

```
:
```

```
}
```

// This is confusing, we can just say it returns

```
0 if s1 == s2
```

```
-1 if s1 < s2
```

```
1 if s1 > s2
```

Eg 2: max is assigned with the value of the expression  
return false else it assigns 1

```
max = (a > b) ? 1 : 0
```

In simple words if (a > b) max = 1 else max = 0